CLEMSON School of COMPUTING

Amortized analysis

Amortized analysis

- In an amortized analysis, we estimate the time required to perform a sequence of operations.

- With amortized analysis, we have the opportunity to show that the average cost of an operation is small, even if the cost of such an operation looks big (according to asymptotic notation)

Amortized analysis

Two techniques

- Accounting method
  - Faster to use, intuitive. Better suited for simple operations
- **Potential method**
  - **More formal and structured. Better suited for involved algorithms**

The potential method

- Instead of representing prepaid work as credit stored with specific objects in the data structure, the ***potential method*** of amortized analysis represents the prepaid work as “potential energy,” or just “potential,” which can be released to pay for future operations.

- A ***potential function*** maps each data structure D_i to a real number, which is the ***potential*** associated with the data structure D_i

?

D_i is the data structure after an operation is performed

Real cost

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Amortized cost

Pontential function

=

Phi

D_{i-1} is the data structure before an operation is performed

ci hat = credits earned from algorithm

ci = real cost of algorithm (like, RAM model or whatever)

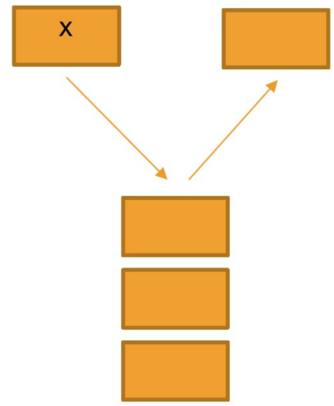
Di = data structure *after* algorithm completed on it

chi(Di) = "potential function", a function that scales with algorithm complexity using some element(s) of data structure (Di) as *input* (not necessarily multiplication)

D (i - 1) = supposedly the data structure before the algorithm is used on it but I don't know why the equation is -1 and not a variable (nvm its because its the step before i but i dont know what i is) (edited)

First example

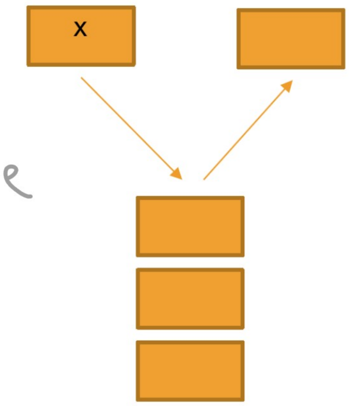
Consider a stack data structure



Step 1 – find the potential function more suitable for our problem.

Idea – the potential function should return a high number when we are close to perform an expensive operation

First example - Stack



ϕ = returns the number of element in the stack

· Thats what determines expensive operations

Cost of function

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Elements after push

PUSH - = 1 + (k'+1) - k' Elements before push
= 2

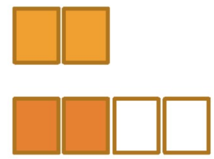
Elements after pop

POP = 1 + (k'-1) – k' Elements before pop
= 0

MULTIPOP = k + (k' - k) - k'
= 0

Second example – Dynamic Table

ϕ = 2*T.num – T.size



0 when T is half empty

T.Num when T is full

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

INSERT - = 1 + (2*(T.num + 1) - T.size) - (2*T.num – T.size)
= 1 + 2 = 3

RESIZE = (T.num+1) + (2*(T.num+1) - 2*T.size) – (2*T.num – T.size)
= T.size +1 + 2T.size + 2 - 2T.size – 2T.size + T.size
= 3

What's next?

- Book chapters
  - Chapter 1.4

- In class activities
  - Use amortized analysis on new examples

- Next class
  - Union find data structures